ESTRATEGIAS DE BÚSQUEDA CON CRITERIOS DE SELECCIÓN QUE DEPENDEN DE TABLAS DE PESOS ASIGNADAS A LOS RECURSOS.

.

Francisco Ibáñez, Daniel Díaz Araya y Raymundo Forradellas

LISI – Laboratorio Integrado de Sistemas Inteligentes
Instituto de Informática - Universidad Nacional de San Juan
e-mail: {fibanez, ddiaz , kike}@iinfo.unsj.edu.ar

Tópicos: Inteligencia Artificial, Sistemas Inteligentes
Keywords: Planning & Scheduling, Constraint Programming, Object Oriented Programming

Resumen

Este trabajo, incluye parte de las estrategias utilizadas para resolver un problema de planificación, en una empresa que produce envases flexibles.
La aplicación involucra por un lado máquinas que pueden modelarse como recursos unarios, y por otro lado, grupos de máquinas, modelables como recursos Discretos.
Para cada recurso y para cada actividad que lo requiera, existe una lista de atributos. Por ejemplo, para el recurso discreto que modela las máquinas huecoimpresoras, los atributos son los siguientes: Línea de Tinta, Tipo de Impresión (Reverso o Superficie), Duración del proceso, Ancho de cilindros, etc.
Para cada máquina huecoimpresora particular (modelada como un Recurso Alternativo), los valores de los atributos mencionados varían en el tiempo, en función de la actividad particular se está procesando.
La aplicación que estamos desarrollando, requiere, para cada máquina particular *m*, que la selección de la actividad a planificarse, en un determinado instante de tiempo *t*, debe seguir criterios que dependen, de los valores de los atributos de la actividad candidata y de los valores de los atributos para la máquina *m* el tiempo *t*. Para el caso de máquinas huecoimpresoras, se debe intentar conservar la línea de tinta y el Tipo de Impresión, y además, se debe procurar planificar actividades consecutivas con distintos anchos de cilindros, e intercambiar actividades de corta y de larga duración. Algunas preferencias son deseables debido a que involucran tiempo de preparación de la máquina para el caso en que no se cumplan, mientras que otras son deseables por razones inherentes a la carga de trabajo de personal. Por ejemplo, si dos actividades planificadas en una misma máquina huecoimpresora utilizan distintas líneas de tinta, es necesario incluir un tiempo de preparación de la máquina, necesario para el cambio de tinta. Por otro lado, es deseable intercambiar actividades de corta y de larga duración a los efectos de distribuir uniformemente la carga de trabajo de los operarios.
Por otro lado, también existen preferencias a la hora de seleccionar una máquina particular de entre las que están disponibles. Por ejemplo, las actividades de corte, tienen un determinado gramaje (cuyo valor depende de los materiales que utiliza). La elección de la máquina más adecuada, depende del gramaje de la actividad a procesar.
Por otro lado, tanto las preferencias relacionadas con la elección de la actividad, como inherentes a la selección de la máquina, tiene asociadas diferentes prioridades.
En este trabajo se presentan los principales algoritmos que hemos utilizado para resolver el problema, y se muestran los resultados obtenidos.

1.- Introducción

Este trabajo, incluye parte de las estrategias utilizadas para resolver un problema de planificación, en una empresa que produce envases flexibles.

Los procesos involucrados en la producción son principalmente: Impresión, Laminado, Corte y Empaque. Según el producto que se desea obtener, estas actividades pueden usar diferentes máquinas.

Cuando se planifican actividades que requieren recursos, existen diferentes formas para asignar recursos a actividades y para planificar las actividades en un tiempo determinado.

Normalmente se elige primero el recurso, y posteriormente se seleccionan las actividades que utilizarán el mismo. Los criterios utilizados, generalmente no incluyen preferencias relacionadas a condiciones que pueden o no cumplirse cuando se planifica una actividad en una determinada máquina.

Para cada recurso, y para las actividades que requieren el recurso, existe una lista de atributos. Por ejemplo, para las máquinas laminadoras (y para las actividades que la requieren), existen atributos tales como la línea de adhesivo. Para cada máquina laminadora particular, las actividades planificadas .

Por otro lado, hay casos en los que una actividad debe elegir una máquina determinada (a partir de un conjunto posible de máquinas), y la conveniencia de usar una u otra depende también de los valores de los atributos de la actividad a planificar.

En este trabajo, se presentan algoritmos que asignan máquinas a actividades, siguiendo criterios relacionados a las características particulares de las máquinas e intentando cumplir condiciones asociadas a tales características.

El modelo utilizado, incluye dos tipos de recursos: unarios y conjunto de recursos alternativos.

Un recurso unario modela una máquina que no puede ser compartida por dos actividades.

Un grupo de n máquinas que tienen las mismas características, y que no sea necesario distinguirlas, normalmente se modelan como recursos discretos de capacidad n.

La salida de una planificación informa que una determinada actividad utiliza una (o más) instancia(s) del recurso discreto, sin especificar cuales son esa instancias concretas.

Típicos problemas de que se modelan como recursos unarios y discretos se encuentran, respectiviamente en [PascalVH,1989]. La resolución de los mismos mediante programación con restricciones se encuentra [Aggoun,1992].

En los casos en los que es necesario agrupar n máquinas con características comunes pero distinguiendo cada una de las instancias, se utiliza un tipo de recurso llamado *conjunto de recursos alternativos*, o simplemente (aunque más confuso) *recursos alternativos*, y cada una de las instancias (que representa cada una de las máquinas) se las denomina *recurso alternativo*.

Una salida, para una actividad que requiere un conjunto de recursos alternativos, debe informar cuales son las instancias (las máquinas particulares) que se han asignado a la misma. En este trabajo utilizamos recursos unarios, recursos alternativas y recursos de tipo conjunto de recursos alternativos, sin embargo, haciendo abuso de terminología, denotaremos estos últimos, simplemente como recursos discretos (de hecho, un recurso de tipo *conjunto de recursos alternativos* puede implementarse utilizando un recurso discreto).

Comencemos considerando el caso simple de un recurso unario donde se desea que se verifiquen tres condiciones *cond1,cond2* y *cond3*. Asumamos además que la condición *cond1* es más importante que *cond2*, y esta más importante que *cond3*. Sean *act1 y act2* dos

actividades candidatas a planificarse en un determinado tiempo en el recurso unario. Supongamos además que *act1* cumple *cond1* y no cumple las restantes, mientras que *act2* no cumple *cond1* pero cumple las condiciones restantes. ¿Cual de las dos actividades conviene planificar?
Por otro lado, si una actividad requiere un recurso de tipo conjunto de recursos alternativos (que de ahora en más referiremos simplemente como discreto), puede existir cierta conveniencia en elegir un recurso alternativo determinado. Para el caso de las máquinas de corte, existen seis máquinas particulares que se modelan cada una de ellas como un recurso alternativo. El criterio para la selección de un recurso alternativo particular, depende del valor del atributo *gramaje*, de la actividad a planificar.
Los algoritmos descriptos permiten incluir criterios para seleccionar los recursos y la actividad a planificar.
La organización del artículo es la siguiente:
Primero se describen diferentes estrategias para seleccionar actividades y recursos. En cada una de ellas se describen las funciones utilizadas, el correspondiente algoritmo, y finalmente se incluyen las críticas.
Posteriormente, se mencionan los principales detalles inherentes a la implementación, y los resultados obtenidos, finalizando con las conclusiones y las líneas futuras de trabajo.

2.- Estrategia 1: recursos unarios, sin prioridades en las condiciones

Primero incluimos sólo recursos unarios. Se asume además, que cada actividad requiere sólo un recurso.
Dado un recurso, para cada atributo del mismo, existe una condición asociada. Por ejemplo, para un recurso que modela máquinas impresoras, al atributo *Línea de tinta* se le asocia la condición *conserva la línea de tinta*, mientras que al atributo *Ancho de cilindro* se le asocia la condición *Ancho de cilindro diferente.*
Para el primer caso, una actividad *a* cumple la condición *conserva la línea de tinta* cuando se planifica en el recurso *r* en un tiempo *t*, si el valor del atributo *Línea de tinta* de la actividad *a* es igual al valor del atributo *Línea de tinta* del recurso *r* en el tiempo *t.*
Para el segundo caso, la condición se cumple si los valores de los atributos (de la actividad y del recurso en el tiempo t) son distintos.
Sean Actividades, RecUnarios y CondRecUnars, conjuntos no vacíos que representan respectivamente: todas las actividades aun no planificadas, todos los recursos unarios, y todas las condiciones.

2.1.- Funciones utilizadas en el algoritmo

ComMin: Actividades $\rightarrow$ N_0
RecUnar: Actividades $\rightarrow$ RecUnarios
CumpleUnar: Actividades X Condiciones $\rightarrow$ {0,1}
CondsUnar: RecUnarios $\rightarrow$ P(CondRecUnars)
ConvUnar: Actividades X P(CondRecUnars) $\rightarrow$ N_0
(N_0 representa el conjunto de los naturales (N) con el cero incluido y P(CondRecUnars) denota el conjunto de partes del conjunto CondRecUnars)

2.2.- Descripción de las funciones

ComMin: Toma como argumento una actividad aun no planificada, y retorna el mínimo tiempo de comienzo en que podría planificarse.

Dada una actividad *act*, el mecanismo utilizado para evaluar *ComMin(act)* garantiza por un lado que retornará un valor (sea *comMin*) y por otro lado que será posible planificar la actividad *act* en el tiempo *comMin*.
RecUnar: toma como argumento una actividad y retorna el recurso unario requerido.
CumpleUnar: toma como argumento una actividad *act* y una condición *cond*, y retorna un 1 si *act* cumple la condición *cond* cuando se planifica en el tiempo *ComMin(act)* en el recurso unario *RecUnar(act)*, caso contrario retorna 0.
CondsUnar: toma como argumento un recurso unario y retorna el conjunto de condiciones asociadas al mismo.
ConvUnar se define en función de las anteriores del siguiente modo:

$$\text{ConvUnar(act, Cond)} = \sum_{\text{cond} \in \text{Cond}} \text{CumpleUnar(act, cond)}$$

con act∈Actividades , r = RecUnar(act), y Cond = CondUnar(r).

2.3.-Algoritmo

1 Min = $_{\text{Minimo act} \in \text{Actividades}}$ ComMin(act)
(Obtiene el tiempo mínimo en que puede planificarse una actividad).

2 Cmin = {act∈Actividades : ComMin(act) = Min}
(Obtiene el conjunto de actividades que tienen mínimo tiempo de comienzo igual a Min)
Notar que Cmin≠∅, debido a que Actividades≠∅.

3 Seleccionar una actividad de Cmin (sea *a*)

4 r = RecUnar(a)
(Obtiene el Recurso requerido por la actividad *a*).

5 Cand = {act∈Cmin : RecUnar(act) = r }
(Obtiene el conjunto de todas las actividades con cota inferior *Min* que requieren el recurso *r*)

6 Obtener el conjunto C del siguiente modo:
Conds = CondsUnar(r),
MaxConv = Máximo $_{\text{act} \in \text{Cand}}$ ConvUnar(act, Conds), y
C = {act∈Cand: ConvUnar(act, Conds) = MaxConv}
Esta estrategia seleccionará una actividad que cumple la mayor cantidad posible de condiciones referidas al recurso *r* (obtenido en el paso 4)

7 Seleccionar una actividad del conjunto C.

En los pasos 3 y 7, para seleccionar una actividad, se tienen en cuenta criterios particulares como por ejemplo aquellos relacionados con las fechas máximas de entrega.

2.4.- Crítica:

1. El criterio de elección no asigna ninguna prioridad a las condiciones.

2. La evaluación del conjunto C no es global, respecto a todos los recursos, sino que la realiza sobre las actividades que requieren el recurso *r* obtenido a partir de la actividad *a* seleccionada en paso 3.

3.- Estrategia 2: recursos unarios, prioridades en las condiciones

La estrategia anterior, tiene en cuenta las condiciones asociadas a los recursos en la planificación de actividades. Sin embargo, asigna las mismas prioridades a cada una de las condiciones. La siguiente estrategia, permite imponer un orden sobre las condiciones, que reflejará la relevancia de las mismas. El algoritmo, tendrá en cuenta este orden en la selección de actividades.
El algoritmo usa las mismas funciones que las utilizadas para la Estrategia 1, salvo la función *CondsUnar*, que en este caso retorna una lista de elementos no repetidos, de la forma $[cond_1, ..., cond_n]$, con $n \leq$ cardinalidad(CondRecUnars) y $cond_i \in$CondRecUnars.$(1 \leq i \leq n)$

3.1.- Algoritmo

Todos los pasos son idénticos a la Estrategia 1, salvo el paso 6, que se reemplaza por el siguiente:

6 Obtener el conjunto C del siguiente modo:

Sea Conds = CondsUnar(r) = $[cond_1, ..., cond_n]$,

$C_1 = \{act \in Cand : cumpleUnar(act, cond_1) = 1\}$
si $C1 \neq \emptyset$ $C = C_1$
caso contrario C = Cand
$C_2 = \{act \in C : cumpleUnar(act, cond_2) = 1\}$
si $C_2 \neq \emptyset$ $C = C_2$
...
$C_n = \{act \in C : cumpleUnar(act, cond_n) = 1\}$
si $C_n \neq \emptyset$ $C = C_n$

3.2.- Crítica:

1. El criterio de elección asigna demasiada importancia a las primeras condiciones respecto a las siguientes. En particular, luego de ejecutar el paso 5, basta encontrar una actividad (en el conjunto *Cand*) que cumpla la primera condición (pudiendo o no cumplir las condiciones restantes), para que queden excluidas de la selección final, todas las actividades (pertenecientes al conjunto *Cand*) que no cumplen la primera condición.
2. La evaluación del conjunto *C* no es global, respecto a todos los recursos, sino que la realiza sobre las actividades que requieren el recurso *r* obtenido a partir de la actividad *a* seleccionada en paso 3.

4.- Estrategia 3: recursos discretos, pesos asignados a las condiciones

Las estrategias anteriores trabajan sólo con recursos unarios. Las siguientes, están desarrolladas para recursos discretos. Para el caso de que sea necesario incluir también recursos unarios, basta considerar recursos discretos formados por un único recurso alternativo.
Otra limitación de las estrategias anteriores, es que constituyen dos extremos:
La Estrategia 1, asume que todas las condiciones (asociadas a un determinado recurso) tienen la misma importancia, mientras que la Estrategia 2, considera que la primera condición es más importante que todas las restantes.

La siguiente estrategia permite generar alternativas intermedias (respecto a las estrategias anteriores) y diferentes para cada recurso, mediante asignaciones de pesos.
Para cada caso particular, y para cada recurso, se deberán asignar pesos a las condiciones asociadas, en relación al grado de importancia de las mismas.

4.1.-Funciones utilizadas en el algoritmo

Sean *RecDiscretos* y *RecAlternativos*, *CondRecDiscrs*, conjuntos que representan respectivamente: todos los recursos discretos, todos los recursos alternativos, y todas las condiciones.
RecDiscr: Actividades $\rightarrow$ RecDiscretos
CumpleAlt: Actividades X RecAlternativos X CondRecDiscrs $\rightarrow$ {0,1}
CondsDiscr: RecDiscretos $\rightarrow$ P(CondRecDiscrs)
P(CondRecDiscrs) denota el conjunto de partes del conjunto CondRecDiscrs
Pos: Actividades X RecAlternativos $\rightarrow$ {0,1}
Peso:CondRecDiscrs $\rightarrow$ N
RecAlt: RecDiscretos $\rightarrow$ P(RecAlternativos).

Tanto a la función *CondsDiscr* como a la función *RecAlt* le imponemos la restricción adicional de que las imágenes de cada uno de los elementos que forman el dominio, constituyen una partición del codominio
La justificación de esta restricción para el caso de la función *RecAlt*, se basa en el hecho de que cada recurso alternativo debe estar asociado a un único recurso discreto.
Para el caso de la función *CondsDiscr* no es estrictamente necesario (pues podría haber una condición como por ejemplo *conservar la línea de tinta* aplicable a dos recursos discretos diferentes) pero es conveniente usar dos identificadores (*conservar la línea de tinta 1*, *conservar la línea de tinta 2*) a los efectos de simplificar el algoritmo. Por ejemplo, sin esta restricción, la función *Peso* requeriría como argumento un recurso discreto (además de una condición). Notar que para la función *CondsUnar* de las estrategias anteriores, no es necesario imponer esta restricción.
ConvAlt: Actividades X RecAlternativos X P(CondRecDiscrs) $\rightarrow$ N0

4.2.- Descripción de las funciones

Dada una actividad *act*, el mecanismo utilizado para evaluar *ComMin(act)* garantizaba que es posible planificar la actividad *act* en el tiempo *comMin = ComMin(act)*. Para el caso de recursos alternativos, esto implica que existe al menos un recurso alternativo disponible, (durante el tiempo en que se procesa *act*) que podrá asignársele a la actividad *act*.
RecDiscr: toma como argumento una actividad, y retorna el recurso discreto requerido.
CumpleAlt: toma como argumentos una actividad *act*, un recurso alternativo *rAlt*, y una condición *cond*, y retorna 1 si *act* cumple la condición *cond* en el tiempo *comMin(act)* en el recurso alternativo *rAlt*. Caso contrario la función retorna 0.
CondsDiscr: toma como argumento un recurso discreto, y retorna el conjunto de condiciones asociadas al mismo.
Pos: toma como argumentos, una actividad *act*, y un recurso alternativo *rAlt*, y retorna 1 si es posible planificar *act* en el tiempo *comMin(act)* en el recurso alternativo *rAlt*. Caso contrario retorna 0.
Peso: Toma como argumento una condición y retorna un valor, que representa el grado de importancia de la misma.

RecAlt: toma como argumento un recurso discreto y retorna el conjunto de recursos alternativos que forman parte del mismo.
Dado un recurso discreto particular *Rec*, un Recurso Alternativo *recAlt* $\in$ *RecAlt(Rec)* y *Conds = CondsDiscr(Rec)*, la función *ConvAlt* se define en función de las anteriores del siguiente modo:

ConvAlt(act, recAlt, Conds) =

$$\text{Pos(act, recAlt)} * \sum_{\text{cond} \in \text{Conds}} \text{CumpleAlt(act, recAlt,cond)*peso(cond)}$$

4.3.- Algoritmo

Los pasos 1 a 3 son los mismos de los algoritmos anteriores.

4 r = RecDiscr(a)
(Obtiene el Recurso requerido por la actividad *a*).

5 Cand = {act$\in$Cmin : RecDiscr(act) = r }
(Obtiene el conjunto de todas las actividades con cota inferior *Min* que requieren el recurso *r*)

6 Conds = CondsDiscr(r),
MaxConv = Máximo act$\in$Cand, recAlt$\in$RecAlt(r) ConvAlt(act, recAlt, Conds)
Pares = {(a, rAlt) : a$\in$Cand, rAlt$\in$RecAlt(r), ConvAlt(a, rAlt, Conds) = MaxConv}
(Obtiene el conjunto de pares act- rec. alternativo, que maximizan la función *ConvAlt*).

7 Seleccionar un elemento del conjunto Pares.

En los pasos 3 y 7 se tienen en cuenta criterios particulares como en los casos anteriores.

4.4.- Características de la estrategia

Las elecciones de los valores para la función *Peso* determinan las estrategias concretas para elegir actividades.
Consideremos el caso simple de un solo recurso *r*.
Sea

CondsDiscr(r) = {cond_1, ..., cond_n}, con n = cardinalidad(CondRecUnars)

Si elegimos

$\text{Peso}(\text{cond}_i) = 2^{i-1}$ $(1 \le i \le n)$

obtendremos una estrategia semejante la Estrategia 2.
Consideremos las siguientes hipótesis para la estrategia 3:
Puesto que estamos considerando que existe un solo recurso (sea *r*), el recurso discreto obtenido en el paso 4 del algoritmo será *r*. Asumamos también que *rAlt* es el único recurso alternativo que forma parte de *r* (conceptualmente *r* se convertiría en recurso unario).
Sea

Cand = {a1, a2}, y sea Conds = CondsDiscr(r) = {cond1, cond2, cond3}.

Asumamos además que

CumpleAlt(a1, rAlt1, cond1) = 1, y
CumpleAlt(a1, rAlt1, cond2) = CumpleAlt(a1, rAlt1, cond3) = 0

Consideremos, por otro lado, las siguientes hipótesis para la Estrategia 2:

Existe un único recurso (unario) *r'*, que será el obtenido el paso 4 del algoritmo.
Sea

Cand = {a1, a2}, y sea Conds' = CondsUnar(r') = [cond1', cond2', cond3'].

Asumamos además que

CumpleUnar(a1, cond1') = 1, y
CumpleUnar(a1, cond2') = CumpleUnar(a1, cond3') = 0

Es fácil probar que ambos algoritmos conducen a resultados equivalentes. Concretamente, el algoritmo de la Estrategia 2 obtiene el conjunto *C = {a1}* el paso 6, mientras que el algoritmo para la Estrategia 3 conduce al conjunto *Pares = {a1, rAlt1}* el paso 6.
Por otro lado, si en la Estrategia 3 consideramos la función *Peso* definida del siguiente modo:

$Peso(cond_i) = c$ $(1 \le i \le n)$, donde *c* es una constante,

obtendremos un resultado equivalente al obtenido con la Estrategia 1. El algoritmo de la Estrategia 1 obtiene el conjunto C = {a2} en el paso 6, mientras que el algoritmo para la Estrategia 3 conduce al conjunto Pares = {a2, rAlt1}.
Si tuviéramos el caso de dos recursos discretos, sería posible asignar a uno de ellos, pesos que conduzcan al comportamiento de la Estrategias1, y al otro, pesos que conduzca a la Estrategia 2. En general, es posible asignar un juego de valores para los pesos, a cada recurso discreto, en función del grado de importancia de las condiciones asociadas a los mismos.

4.5.- Crítica:

1 La evaluación del conjunto *Pares* no es global, respecto a todos los recursos, sino que la realiza sobre las actividades que requieren el recurso *r* obtenido a partir de la actividad seleccionada *a* en paso 3.

5.- Estrategia 4: extensión para considerar todos los recursos.

Cada vez que se va a seleccionar una actividad en el algoritmo anterior, se tiene en cuenta sólo un recurso discreto (aquel obtenido en el paso 4). La siguiente estrategia, tendrá en cuenta todos los recursos discretos requeridos por las actividades seleccionadas en el paso 2.

5.1.- Funciones

ConvDiscr: Actividades X RecDiscretos $\rightarrow$ N0
ConvAct: Actividades $\rightarrow$ N0

5.2.- Descripción de las funciones

ConvDiscr(act, recDiscr) =

Maximo $_{recAlt \in RecAlt}$(recDiscr) ConvAlt(act, recAlt, Condiciones(recDiscr))
ConvAct(act) = ConvDiscr(act, RecDiscr(act)),

5.3.- Algoritmo

Los pasos 1 y 2 son los mismos de los algoritmos anteriores.

3 MaxConvGlobal = Máximo $_{act \in Cmin}$ ConvAct(act)

4 Pares = {(a, rAlt) : a$\in$Cmin, rDiscr = RecDiscr(a), rAlt$\in$RecAlt(rDiscr),
Conds = CondsDiscr(rDiscr) ,
ConvAlt(a, rAlt, Conds) = MaxConvGlobal }

(Obtiene el conjunto de pares actividad - recurso Alternativo, que maximizan la función ConvAlt).

5 Seleccionar un elemento del conjunto Pares.

Con esta estrategia, sólo en el paso 5 se tienen en cuenta criterios particulares.

5.4.- Crítica:

No tiene en cuenta ninguna preferencia respecto a los recursos alternativos.

6.- Estrategia 5: preferencias de recursos alternativos

En esta sección, se extiende la estrategia descripta anteriormente, al caso en que también existan preferencias inherentes a los recursos alternativos.

6.1.- Funciones

PrefRecAlt: Actividades X RecAlternativos $\rightarrow$ N
ConvAlt': Actividades X RecAlternativos X P(CondRecDiscrs) $\rightarrow$ N0

6.2.- Descripción de las funciones

PrefRecAlt: toma como argumentos, una actividad y un recurso alternativo, y retorna un número natural, cuyo valor se establece en función de la conveniencia de asignar el recurso alternativo a la actividad. Por ejemplo, para el caso de las máquinas de corte, mencionado anteriormente, el valor de la función se determina en base al gramaje asociado a la actividad (primer argumento) y al grado de preferencia de utilizar la máquina representada por el recurso alternativo (segundo argumento) para ese gramaje.
ConvAlt'(a, rAlt, Conds) =

$$Pos(a, rAlt) * (PrefRecAlt(a, rAlt) + \sum_{c \in Conds} CumpleAlt(a, rAlt,c)*peso(c))$$

6.3.- Algoritmo

El algoritmo para la Estrqategia 5, se obtiene reemplazando la función *ConvAlt* en el algoritmo descripto anteriormente, por *ConvAlt'*.

6.4.- Características de la estrategia

Diferentes valores para la función *Peso* y para la función *PrefRecAlt* determinan diferentes estrategias concretas para elegir una actividad.
Consideremos nuevamente el caso simple de un único recurso *r*.
Asumamos que *r* está compuesto por *m* recursos alternativos: $rAlt_1$, ..., $rAlt_m$.
Sea

$$CondsDiscr(Rec) = \{cond_1, ..., cond_n\},$$

si elegimos

$$Peso(cond_i) = m*2^{i-1} \text{ y}$$

$$PrefRecAlt(act, recAlt_j) = j \quad (1 \leq j \leq m),$$

obtendremos una estrategia que considera más importante las condiciones asociadas al recurso r, que las preferencias de los recursos alternativos. De hecho todas las segundas componentes de los pares *(act, rAlt)* del conjunto Pares, tienen la propiedad de ser iguales.

Formalmente,

(1) si (act, rAlt)$\in$Pares y (act', rAlt')$\in$Pares $\Rightarrow$ rAlt = rAlt'.

Por otro lado, las condiciones que cumplen todas las actividades que constituyen un par del conjunto Pares (con el recurso alternativo único), son exactamente las mismas, y por lo tanto:

(2) si (act, rAlt)$\in$Pares y (act', rAlt')$\in$Pares $\Rightarrow$

cumpleAlt(act, rAlt, condi) = cumpleAlt(act', rAlt', condi) $(1 \leq i \leq n)$

Consecuentemente, podrían haberse encontrado primero las condiciones que debe cumplir una actividad a seleccionarse, desconsiderando totalmente la función *PrefRecAlt*, y en una etapa posterior encontrar el recurso alternativo único.

Las demostraciones de (1) y (2), están disponibles, pero no las incluimos debido a que son muy extensas.

A diferencia del caso anterior, si asignamos a la función *PrefRecAlt*, valores mayores que a la función *Peso*, estableceremos más importancia a la selección de los recursos alternativos que a las condiciones.

Lo más interesante, desde el punto de vista práctico, es que para cada recurso discreto, se pueden asignar juegos de pesos distintos (para las condiciones y para la elección de los recursos), generando estrategias especializadas para cada caso.

7.- Implementación

La aplicación está implementada en C++, empleando rutinas de ILOG SCHEDULER [I,Sch-R,99] y de ILOG SOLVER [I,Sol-R,99].

Los algoritmos presentados son susceptibles a ser implementados en lenguajes que incluyan Constraint Solvers (CS) en dominios finitos, como por ejemplo [PascalVH,1989], [OZ,2000]. La posibilidad de poder definir CSs facilita la implementación de estos algoritmos, debido a que es necesario generar un comportamiento complejo en el manejo de las restricciones inherentes a las actividades y a los recursos. Lenguajes como [ECL^{i}PS^e,1995] incluyen la posibilidad de generar CS mediante reglas de manejo de restricciones [Fruhw, 93].

Otro posibles enfoque es utilizar algoritmos que permitan integrar restricciones temporales métricas y simbólicas (para aumentar la propagación y reducir el árbol de búsqueda) [Ibañez, 93] o instancias de CLP(X) [Jaffar, 87] que incluyan las actividades como primitivas del lenguaje e incluyen un CS especializado para problemas de razonamiento temporal [Ibañez,96].

OPL (Optimization Programming Language) [OPL,1999] es una alternativa muy atractiva, debido a que es muy poderoso y muy declarativo, pero preferimos realizar la implementación en ILOG, principalmente debido a la potencia que este presenta en el manejo de las primitivas relacionadas con la tabla de tiempo.

Un análisis comparativo de diferentes los diferentes enfoques basados en restricciones para dominios finitos puede encontrarse en [Fernand,2000].

Debido a que es necesario conocer los valores de los atributos de cada recurso alternativo, en cualquier instante de tiempo, fue necesario implementar nuestras propias tablas de tiempo, a los efectos de incluir el manejo de esta información. No es posible reutilizar código de ILOG, inherente a las tablas de tiempo, debido a que sería necesario manipular datos miembros privados de las clases manejadoras utilizadas para las tablas de tiempos.

Una tabla de tiempo para un recurso alternativo, puede implementarse en C++ mediante un contenedor de objetos de un tipo *T* [Strou, 97]. Cada uno de los objetos representa un intervalo de disponibilidad del mismo. Utilizando contenedores no es necesario programar las inserciones, búsquedas, supresiones, etc.

Los mínimos datos necesarios para representar un intervalo de disponibilidad son el comienzo y el fin del mismo. Sin embargo, para asociarle a cada intervalo de disponibilidad un conjunto de atributos, es necesario conocer el recurso alternativo particular. Todos los recursos alternativo que forman parte del recurso que los agrupa, comparten los mismos atributos, pero diferentes recursos discretos tienen distintos atributos.

La solución adoptada fue definir una clase *intervDisponibilidad* con sólo dos datos miembros que representan el inicio y el fin del intervalo de disponibilidad y definir, para cada recurso discreto, una clase, heredada de *intervDisponibilidad*, que incluye como datos miembros aquellos necesarios para representar los atributos particulares de recurso discreto.

Hay por lo tanto tantas clases (derivadas de *intervDisponibilidad*) como recursos discretos existan.

Cada recurso alternativo será una instancia de un contenedor. Si recAlt1 forma parte del recurso discreto recDiscr1, el tipo de los objetos que constituyen el contenedor para *recAlt*, debe ser la clase (derivada) definida para *recDiscr*, sea *ClaseDiscr1*.
Para definir tablas de tiempos para recursos alternativos que forman parte de diferentes recursos alternativos, utilizamos el concepto de patrones (templates) de C++, y definimos una clase *TablaDeTiemposConAtributos* con la siguiente cabecera:

```
template <class Generica> class TablaDeTiemposConAtributos: public vector<Generica>
```

Para generar una tabla de tiempos para el recurso alternativo *recAlt1*, creamos la instancia *tablaTpo_recAlt1* instanciando *TablaDeTiemposConAtributos* con la clase *ClaseDiscr1*.

```
TablaDeTiemposConAtributos<ClaseDiscr1>  tablaTpo_recAlt1(n);
```

donde *n* es el tamaño inicial del contenedor (ampliable mediante la función predefinida *resize*).

8.- Resultados obtenidos

Sólo se han implementado las Estrategias 1 y 5. El programa que implementa la Estrategia 1 es relativamente veloz, pero no es suficientemente flexible como para reflejar las necesidades de la empresa. En ambas implementaciones, la ejecución se detiene inmediatamente después de obtener la primera solución, debido a la cantidad de actividades y de restricciones que maneja. Una entrada tipo puede tener hasta 1500 actividades, y aproximadamente 6000 restricciones, consecuentemente, es necesario extremar esfuerzos para lograr que la primera solución que encuentra sea aceptable. En esta aplicación, no utilizamos una función objetivo a minimizar, sino que proveemos distintas medidas para evaluar la calidad de los resultados. Entre estas medidas están, el porcentaje de condiciones que se cumplen, y medidas relativas a la fecha de entrega de los pedidos. La Estrategia 5, es suficientemente flexible, pero el tiempo de ejecución aumenta en general, a más del triple.

Es difícil obtener un comportamiento promedio, tanto en tiempo de ejecución como en porcentaje de condiciones que se cumplen, debido a que la salida es fuertemente dependiente de los datos de entrada. Con ambas estrategias surgieron dos problemas para algunos juegos de entrada particulares: no producía respuesta en tiempos aceptables, y por otro lado, pese a aumentar considerablemente el porcentaje de condiciones cumplidas, algunas fechas de entregas no podían respetarse. La razón de este último inconveniente, radica en siguiente hecho. Inicialmente, las actividades se ordenan en función de las fechas de entrega. Bajo la ausencia de manejo de condiciones, las actividades tienden a planificarse en un orden relativamente parecido al original. Al imponer restricciones sobre condiciones, para algunas entradas ocurre que actividades pertenecientes a pedidos con fechas de entrega temprana, van postergándose, debido a que no cumplen suficientes condiciones.

La solución adoptada para resolver ambos problemas fue particionar el conjunto de actividades. Cada partición considera las condiciones sin relación a la partición anterior y consecuentemente originan la posibilidad de que se planifiquen actividades que no cumplen suficientes condiciones.

Las tablas de tiempo de los recursos perduran a lo largo de toda la ejecución, pero la memoria requerida para almacenar los datos inherentes a las actividades y a las restricciones se libera cada vez que termina un segmento (salvo aquella requerida para almacenar los resultados).

Tanto el tamaño de memoria principal requerida como la cantidad de restricciones que necesita manejar en cada segmento, disminuye en forma proporcional al tamaño del segmento, mientras que el tiempo de ejecución se reduce drásticamente conforme disminuye el tamaño del segmento, debido a la naturaleza NP del problema . Por ejemplo, con 1500 actividades, para un sólo segmento, la cantidad de restricciones es igual a 6586, la memoria consumida es 7949020 bytes y el tiempo de ejecución es 22.79 minutos, mientras que particionando en 5 segmentos, los correspondientes valores son los siguientes: 1572 restricciones, 1108996 bytes, 30 segundos.

Pese a las ventajas mencionadas anteriormente, la segmentación soluciona introduce una degradación de la calidad de la solución, respecto a la utilización de las máquinas, debido a que actividades que se planifican en las primeras particiones originan períodos de inactividad de las máquinas ("huecos").

Las actividades pertenecientes a segmentos posteriores, se planifican utilizando estos períodos de inactividad, pero se desperdicia la diferencia entre la duración del mismo y de la actividad. Tamaños de segmentos muy pequeños, degradan demasiado la utilización de las máquinas y conducen, por este motivo, a la violación de las fechas de entrega

El tamaño de la partición es parametrizable, y de este modo, el usuario puede buscar los valores para las condiciones y el tamaño de la partición que le proporciones mejores resultados.

9.- Conclusión y líneas futuras

En este trabajo se han presentado algoritmos que contribuyen a la planificación temporal de actividades, en un problema planteado por una empresa que produce envases flexibles.

Los algoritmos descriptos, permiten seleccionar una actividad y una la máquina particular que se le asignará a la misma, en base a la evaluación de verificación de condiciones que dependen de las características de las máquinas.

Los primeros algoritmos propuestos solucionan parcialmente el problema pero no se adaptan demasiado a las características particulares de las máquinas de la aplicación real.

En los últimos algoritmos, es posible asignar pesos, tanto a las condiciones definidas para cada recurso discreto, como a los recursos alternativos que forman parte del recurso discreto.

Es posible por lo tanto, generar distintas prioridades para las condiciones (que se tendrán en cuenta para seleccionar la actividad), como para los recursos alternativos (que se tendrán en cuenta para seleccionar la máquina particular que utilizará la actividad). De este modo, cada conjuntos de asignaciones de peso para un determinado recurso discreto, genera una política, particular a ese recurso, para elegir una actividad y máquina en el momento de la planificación, pero además, es posible generar distintos conjuntos de asignaciones para cada recurso discreto, de modo tal que cada uno de ellos genera una política, adaptada a las características del recurso discreto particular.

Si bien la aplicación de los algoritmos conducen a planificaciones que tienden a cumplir las condiciones establecidas, el costo que trae aparejado se refleja en términos de dos aspectos prácticos cruciales:

- dejan de respetarse las fechas de entrega
- aumenta el tiempo de ejecución del programa

La segmentación soluciona parcialmente estos inconvenientes pero es necesario ser cuidadoso en la elección del tamaño de segmento para cada entrada particular.

En cuanto a las líneas futuras de trabajo, actualmente estamos trabajando mayoritariamente en aquellas relacionadas con aumentar tanto como sea posible la adaptación del programa a los requerimientos del problema real, pero especialmente en la adaptación de estos algoritmos al tratamiento dinámico de los tiempos de preparación de las máquinas.

La versión que implementa la última estrategia, incluye varias características necesarias para representar el problema real, tales como el uso de calendarios asociados a cada una de las máquinas para determinar los períodos de disponibilidad inicial y la consecuente necesidad de trabajar con actividades interrumpibles, otras inherentes al tratamiento de restricciones temporales complejas entre los puntos de inicio y fin de las actividades, etc.

Los algoritmos presentados permitieron incluir todas estas características, pero no admite el tratamiento de tiempos de preparación de las máquinas dinámico. En la versión actual, los tiempos de preparación de las máquinas sólo intervienen en el cálculo de la duración de la actividad, y se puede conocer antes de comenzar la planificación. Para la mayoría de los casos este tratamiento de los tiempos de preparación de las máquinas es suficiente, debido a que no es necesario modelar la cantidad de producto semielaborado, producido cada vez que finaliza una actividad. Incluso para algunos casos en que el tiempo de preparación de las máquinas se determina dinámicamente mientras se realiza la planificación, en la práctica puede despreciarse. Este es el caso de los cambios de tintas en las máquinas impresoras. Si bien el tiempo de preparación de la máquina (originado por el cambio de tintas) puede ser o no cero (dependiendo de la necesidad de cambiar tintas), en la práctica puede considerarse cero. Sin embargo en otros casos, como por ejemplo en el tipo de impresión (de reverso o de superficie), el tiempo necesario para cambiar de reverso a superficie (o viceversa) toma aproximadamente una hora y es necesario considerarlo. Existe una versión del programa que incluye el tratamiento de preparación de las máquinas dinámico, pero sin aplicar los algoritmos presentados. El próximo paso es extender los algoritmos para tratar apropiadamente con el manejo de preparación de las máquinas dinámico.

10.- Referencias

[Aggoun,1992] A. .Aggoun and N. Beldiceanu. Extending CHIP to solve Complex Scheduling and Packing Problems. In Journe´es Francophones de Programmation Logique , Lille, France , 1992

[Ddiaz,98] Daniel Diaz Araya - Aplicación de Tecnologías Basadas en Restricciones a la Resolución de Problemas Industriales -Tesis de Grado, UNSJ, 1998.

[ECL*i*PS*e*,1995] ECL*i*PS*e* 3.5, User Manual. (1995) . ECRC, Munich.

[Fernand,2000] Antonio Fernandez, Patricia Hill, "A Comparative Study of Eight Constraint Programming Languages over the Boolean and Finite Domains", Kluwer Academic Publishers, Boston, 2000.

[Fruhw, 93] T. Fruhwirth and P. Handschke, Terminological Reasoning with Constraint Handling Rules". PPCP. Workshop, 1993

[Ibañez, 93] F. Ibáñez, Integración de Metric and Symbolic Temporal Constraint ine CLP", Joint Workshop on Spatial and Temporal Databases, Coniston, England, 1993.

[Ibañez,94] F. Ibañez, "CLP(Temp), Integración de Restricciones Temporales Métricas y Simbólicas, en el Marco CLP", Tesis Doctoral, UPValencia, España, 1994.

[Ibañez,96], F. Ibañez, R. Forradellas, L. Rueda, "An Instance of the CLP(X) Scheme which allows to deal with Temporal Reasoning Problems", 3rd Workshop on Logic, Language, Information and Computation (WoLLIC'96), Salvador (Bahia), Brasil, 1996.

[I,Sol-R,99] "Ilog Solver - Reference Manual Version 4.4", Ilog, France, 1999.

[I,Sch-R,99] "Ilog Schedule- Reference Manual Version 4.4", Ilog, France, 1999.

[Jaffar, 87] J. Jaffar and J-L. Lazzes and M.J. Maher, "A Logic Programming Language Scheme", in Logic Programming: Relations, Functions and Equations, Prentice-Hall, pp. 441-467, 1986.

[OPL,1999] P. Van Hentenryck The OPL Optimization Programming Language. The MIT Press. Cambridge, MA,1999.

[OZ,2000] Mozart Documentation http://www.mozart-oz.org/documentation/

[PascalVH,1989] P. Van Hentenryck. Constraint Satisfaction in Logic Programming. Logic Programming Series, The MIT Press. Cambridge, MA,1989.

[Rueda,95] Rueda L. Klenzi R. Gutierrez L. Ibañez F. Forradellas R., "Tratamiento de Problemas de Combinatoria Discreta mediante el Paradigma CLP", 2das. Jornadas de Inteligencia Artificial, Universidad Nacional del Sur, Bahía Blanca, 1995.

[Strou, 97] Bjarne Stroustrup, "C++ Programming languajge Third edition", Addison Wesley, 1997.